Using Spoofax to Support Online Code Navigation

Peter D. Mosses
Delft University of Technology, The Netherlands
Swansea University, UK

Abstract

Spoofax is a language workbench. A Spoofax language specification generally includes name resolution: the analysis of bindings between definitions and references. When browsing code in the specified language using Spoofax, the bindings appear as hyperlinks, supporting precise name-based code navigation. However, Spoofax cannot be used for browsing code in online repositories.

This paper is about a toolchain that uses Spoofax to generate hyperlinked twins of code repositories. These generated artefacts support the same precise code navigation as Spoofax, and can be browsed online. The technique has been prototyped on the CBS (Component-Based Semantics) specification language developed by the PLanCompS project, but could be used on any language after specifying its name resolution in Spoofax.

2012 ACM Subject Classification Information systems → Web interfaces; Software and its engineering → Integrated and visual development environments; Software and its engineering → Development frameworks and environments

Keywords and phrases Spoofax language workbench, name resolution, precise code navigation

Digital Object Identifier 10.4230/OASIcs.EVCS.2023.21

Funding The work reported here is part of the PLanCompS project, which was funded by an EPSRC grant to Swansea University from 2011 to 2015 (EP/I032495/1) in collaboration with colleagues at Royal Holloway University of London and City University in London.

Acknowledgements Eelco Visser (1966–2022) was a visionary leader. For more than two decades, I have benefited greatly from reading his persuasive publications, listening to his animated presentations, and discussing topics of common interest with him – especially modular language specification. Since 2012, I have also been an enthusiastic user of the Spoofax language workbench, developed by Eelco and his Programming Languages group at TU Delft. This paper reports on some work that I carried out as a visitor at TU Delft since 2016. I am grateful to the reviewers of a previous version for helpful comments and suggestions.

1 Introduction

Programming languages (including domain-specific languages (DSLs)) usually allow definitions of named entities, and references to entities via their names.

1.1 Code Navigation

In a GitHub Blog post [4], Douglas Creager explains the concept of *code navigation* based on names:

> Code navigation is a family of features that let you explore the relationships in your code and its dependencies at a deep level. The most basic code navigation features are "jump to definition" and "find all references." Both build on the fact that *names* are pervasive in the code that we write.

Although "jump to definition" sounds simple enough, the difficulty of its specification and implementation depends highly on the code language. The determination of the bindings between definitions and references is called name resolution; different languages often have quite different rules for it.

Name resolution can significantly complicate searching for the definition of a particular name when using an ordinary browser or editor. For example, a language may allow references to a name from other files than the file containing the definition; it may also allow the same name to be defined more than once, possibly with shadowing. For such languages, simple project-wide searches for the definition of a specific name, or for all references to that name, may be imprecise, and return unhelpful false positives.

Eelco Visser led the development of three declarative meta-languages for specifying name resolution: NaBL [11], NaBL2 [19, 25], and Statix [20, 22, 26, 27]. The Spoofax language workbench [8, 10, 31] implements name resolution for any language whose rules are specified in one of these meta-languages. When browsing a file in the specified language, Spoofax equips each reference to a defined name with a hyperlink; clicking on the hyperlink moves the cursor directly to the referenced definition. If the definition is in a different file from the reference, Spoofax automatically opens an editing window on that file. Similarly, Spoofax equips each definition with the list of hyperlinks to all the current references to it. These hyperlinks are precise, and provide reliable support for code navigation.

The languages whose name resolution rules have been specified and implemented in Spoofax include:

- the main Spoofax meta-languages: SDF2, SDF3, Stratego, NaBL, NabL2, Statix, DynSem, Dynamix;
- DSLs used in the production of various software systems: currently WebDSL, IceDust, Green-Marl, PGQL, and LeQuest;
- languages used in Computer Science courses: MiniJava, Jasmin, Tiger, PAPL;
- demonstration languages provided in MetaBorg [14]: SIMPL, QL/QLS, Grace, a subset of Go, MetaC, and Pascal; and
- specification languages developed for other purposes, such as CBS [17] (a meta-language for component-based semantics).

Spoofax users can exploit the hyperlinks between definitions and references to navigate local clones of code repositories in the above languages; but when accessing the same repositories online, those hyperlinks are not available.

1.2 Browsing online code repositories

Suppose that we find a code repository online, and we would like to browse it with precise name-based code navigation using the Spoofax language workbench. Spoofax can only be used to browse local files, so we need to download or clone the repository. We also need to find and download a Spoofax language project for each language used in the code repository (except for the main Spoofax meta-languages). After building those language projects in Spoofax, we can finally browse the cloned repository with precise code navigation.

The necessity of downloading the code repository and the required language projects surely discourages browsing. Moreover, the Spoofax language workbench is currently available only for use in Eclipse and IntelliJ IDEA; users not familiar with either of those IDEs may be reluctant to install them just for browsing some code repository. And users behind company firewalls might not even be allowed to install such 3rd-party software as Spoofax.

Ideally, an online code repository would support precise name-based navigation in standard web browsers, *without* the need to download any files or install new software. The rest of this paper explains how to do that, using Spoofax itself:

- Section 2 gives an overview of a repository that supports precise name-based navigation from references to definitions in web browsers.

- Section 3 presents the Spoofax language project that created the same name-based navigation for local use.
- Section 4 explains how that navigation is made available in web browsers, using a toolchain that combines Spoofax with some standard applications to create "hyperlinked twins" of unlinked code repositories.
- Section 5 relates the Spoofax-based approach to some other approaches that provide name-based code navigation in web browsers.
- Section 6 concludes with suggestions for further development of the presented approach.

2 The CBS-beta Repository

CBS [1] is a meta-language for component-based semantics, developed in the PLanCompS project [21]. See previous publications [3, 17, 28, 29] for motivation, foundations, and explanations of CBS. The CBS-beta repository provides a beta-release of language specifications using CBS. It has two main parts

Funcons-beta: A proposal for an initial library of so-called fundamental programming constructs (funcons).

Languages-beta: Examples of language specifications based on the funcons in Funcons-beta.

(Two further parts are marked as "unstable", and still being developed.)

2.1 CBS specifications

Funcons [18] are *reusable* components: the same funcon can be used, unchanged, in the specifications of many different languages. Funcons correspond closely to concepts of high-level programming languages such as data and control flow, scopes of bindings, mutable variables, streams, abrupt termination, procedural abstraction, etc.

A funcon definition declares the name of the funcon, and specifies its signature: the types of its arguments (if any), and of the value(s) that it returns. The definition also specifies small-step operational semantics rules for evaluating the funcon.

Funcon names are strongly suggestive of the corresponding concepts. The Funcons-beta library defines about 400 funcons. Many of the funcon names are quite long, but have short aliases (e.g., "allocate-initialised-variable" has the alias "alloc-init").

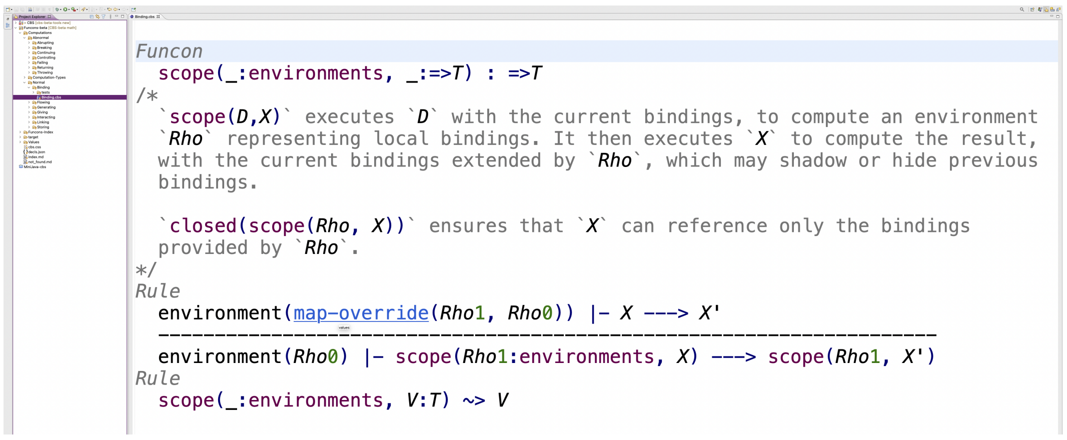
Funcons are to have *fixed* definitions, so *no version control* will be needed for their safe reuse in CBS language specifications. Crucially, adding new funcons does not require any changes to the definitions of existing funcons, thanks to the use of a modular variant of structural operational semantics (MSOS) [15] in CBS.

Apart from illustrating the use of CBS, the aim of the beta-release of CBS and its initial library of funcons was to allow a public review of the definitions, and subsequent adoption of suggestions for improvement, before their finalisation in a full release. The funcon definitions in Funcons-beta have all been validated by empirical testing. The Funcons-beta library has also been found useful in the iCoLa meta-language for incremental (meta-)programming [5].

A language specification in CBS resembles a conventional denotational semantics: it defines the language syntax by a context-free grammar, and it defines semantic functions – mapping phrases in the language to their denotations – by equations. In CBS, the denotations are simply compositions of funcons, instead of higher-order functions.

Languages-beta provides the beta-release of five examples of language specifications in CBS, based on the funcon definitions in Funcons-beta. The examples range from a small introductory example to a substantial sub-language of OCaml.

Table 1 Browsing a local clone of the CBS-beta repository in Spoofax.



```
Funcon
  scope(_:environments, _:=>T) : =>T
/*
  `scope(D,X)` executes `D` with the current bindings, to compute an environment
  `Rho` representing local bindings. It then executes `X` to compute the result,
  with the current bindings extended by `Rho`, which may shadow or hide previous
  bindings.

  `closed(scope(Rho, X))` ensures that `X` can reference only the bindings
  provided by `Rho`.
*/
Rule
  environment(map-override(Rho1, Rho0)) |- X ---> X'
  --------------------------------------------------------------------
  environment(Rho0) |- scope(Rho1:environments, X) ---> scope(Rho1, X')
Rule
  scope(_:environments, V:T) ~> V
```

Each language specification is independent, and (implicitly) imports all the funcons that it references. Regarding name resolution, the Funcons-beta library corresponds to a single module or package, and users do not need to be aware of the internal file structure of the library.

When browsing funcon definitions and language specifications, precise name-based navigation from references to definitions is essential. Section 3 presents a Spoofax language project for CBS that supports such navigation when using Spoofax to browse a local clone of CBS-beta. Table 1 shows how the definition of the funcon "`scope`" looks in Spoofax. Apart from the first (defining) occurrence of "`scope`", all the names in it are references, equipped with hyperlinks to the respective definitions.

2.2 Browsing CBS-beta online

The CBS sources of the specifications in CBS-beta are available in a repository on GitHub. Table 2 shows how the same definition as in Table 1 looks when browsing the CBS sources on GitHub. The text has no hyperlinks, with no support at all for name-based navigation.

However, the CBS-beta repository also contains the sources of an associated website, which includes *hyperlinked twins* of each CBS source file. This website is served by GitHub Pages at `https://plancomps.github.io/CBS-beta/`.

The hyperlinked twins of a CBS source file are in the following formats.

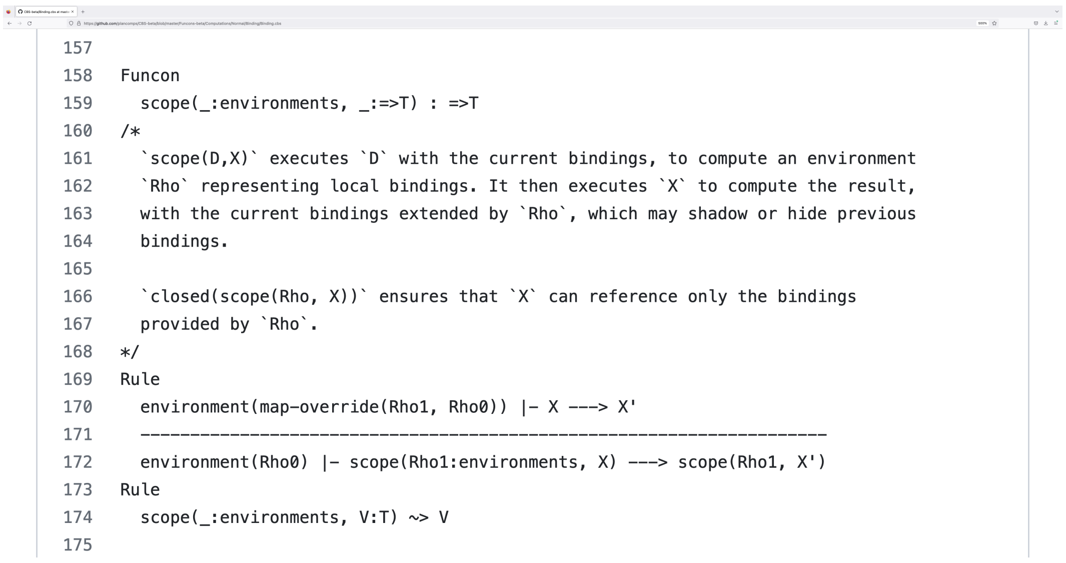
PLAIN: In this format, the web page displays a verbatim copy[1] of the source file. Table 3 shows how the same definition as before looks in the PLAIN format. Names are highlighted: different colours distinguish between names of funcons, syntax sorts, semantic functions, and meta-variables. References are hyperlinked to declarations.

PRETTY: In this format, the web page displays CBS with mathematical typography (as in published articles about CBS). Table 4 shows how the same definition as before looks in the PRETTY format. As in the PLAIN format, names are highlighted, and references are hyperlinked to declarations.

PDF: This format is simply a PDF rendering of the PRETTY web page. Table 5 shows how the same definition as before looks in the PDF format.

[1] The PLAIN format deviates from a verbatim copy by using different font styles.

Table 2 Browsing the CBS-beta repository on GitHub.



CBS-beta/Binding.cbs at master
https://github.com/plancomps/CBS-beta/blob/master/Funcons-beta/Computations/Normal/Binding/Binding.cbs

```

Funcon
  scope(_:environments, _:=>T) : =>T
/*
  `scope(D,X)` executes `D` with the current bindings, to compute an environment
  `Rho` representing local bindings. It then executes `X` to compute the result,
  with the current bindings extended by `Rho`, which may shadow or hide previous
  bindings.

  `closed(scope(Rho, X))` ensures that `X` can reference only the bindings
  provided by `Rho`.
*/
Rule
  environment(map-override(Rho1, Rho0)) |- X ---> X'
  ----------------------------------------------------------------------
  environment(Rho0) |- scope(Rho1:environments, X) ---> scope(Rho1, X')
Rule
  scope(_:environments, V:T) ~> V

```

The PLAIN and PRETTY hyperlinked twins of a CBS source file include links to each other, to the PDF twin, and to the source file on GitHub.

CBS specifications may include informal text as comments, with embedded references to names. In the source files, comments are enclosed in `/*...*/`, and embedded references in back-ticks; in the hyperlinked twins, the comments are displayed as running text, and the formal CBS specifications are displayed as code blocks.

Section 4 explains how Spoofax is used to generate the hyperlinked twins from the CBS source files.

3 A Spoofax Language Project for CBS

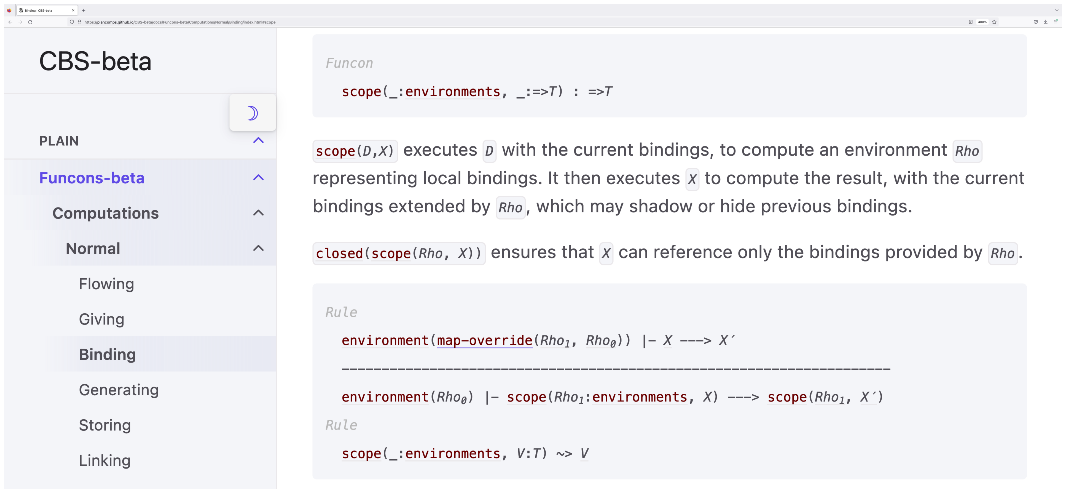
The author has developed a Spoofax language project for CBS. It specifies the syntax of CBS using the meta-language SDF3. Spoofax generates a parser from the SDF3 specification. When a CBS source file is opened, Spoofax creates an editor window for the text, and automatically re-parses the text each time it is edited.

Spoofax uses syntax highlighting to display well-formed phrases; when an edit introduces an error, Spoofax moves the cursor to the source of the error. Spoofax also warns about any phrases that have ambiguous parses.

The syntax of CBS is simpler than that of typical high-level programming languages, and its specification in SDF3 was reasonably straightforward. Unusually, comments are included in ASTs, and formal terms can be embedded in comments (enclosed in back-ticks, as in Markdown).

The CBS language project specifies name resolution using the meta-language NaBL2 [19, 25]. Name resolution in CBS involves checking that all referenced names of each sort (syntax, semantics, funcons) are declared uniquely. Type analysis checks that semantic functions are applied only to the sort of syntax argument specified in their signatures, and that funcons are applied only to the expected number of arguments. However, the analysis of a CBS specification involves the analysis of the entire funcons library, and the implementation of NaBL2 is non-incremental, so re-analysis needs to be suspended while editing even a small CBS source file. (See Section 6 for how to address this drawback.)

Table 3 Browsing a PLAIN hyperlinked twin.



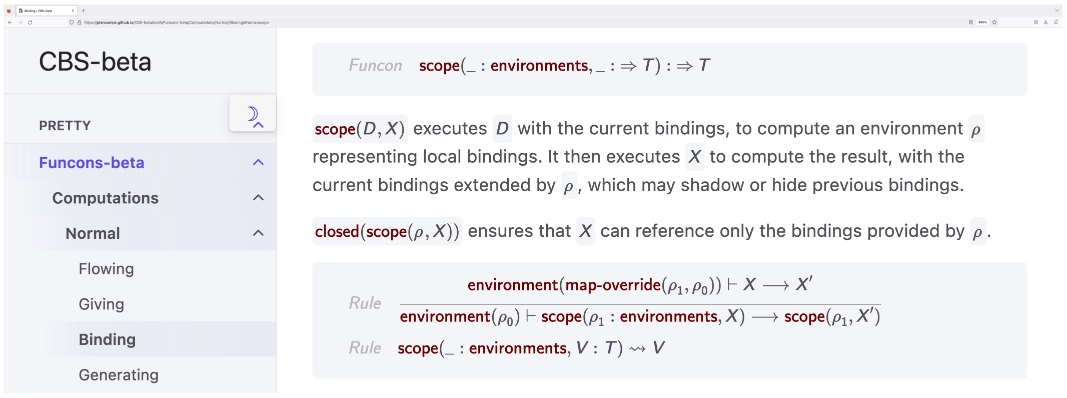
```
Funcon
  scope(_:environments, _:=>T) : =>T
```

`scope(D,X)` executes `D` with the current bindings, to compute an environment `Rho` representing local bindings. It then executes `X` to compute the result, with the current bindings extended by `Rho`, which may shadow or hide previous bindings.

`closed(scope(Rho, X))` ensures that `X` can reference only the bindings provided by `Rho`.

```
Rule
  environment(map-override(Rho1, Rho0)) |- X ---> X′
  -----------------------------------------------------------------
  environment(Rho0) |- scope(Rho1:environments, X) ---> scope(Rho1, X′)
Rule
  scope(_:environments, V:T) ~> V
```

Table 4 Browsing a PRETTY hyperlinked twin.



Funcon $\text{scope}(\_ : \text{environments}, \_ : \Rightarrow T) : \Rightarrow T$

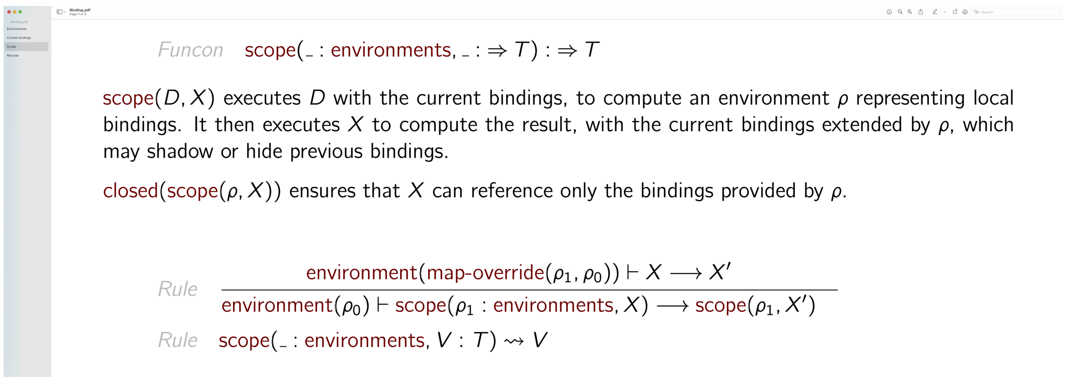
$\text{scope}(D, X)$ executes D with the current bindings, to compute an environment ρ representing local bindings. It then executes X to compute the result, with the current bindings extended by ρ, which may shadow or hide previous bindings.

$\text{closed}(\text{scope}(\rho, X))$ ensures that X can reference only the bindings provided by ρ.

Rule $$\frac{\text{environment}(\text{map-override}(\rho_1, \rho_0)) \vdash X \longrightarrow X'}{\text{environment}(\rho_0) \vdash \text{scope}(\rho_1 : \text{environments}, X) \longrightarrow \text{scope}(\rho_1, X')}$$

Rule $\text{scope}(\_ : \text{environments}, V : T) \rightsquigarrow V$

Table 5 Browsing a PDF hyperlinked twin.



Funcon $\text{scope}(\_ : \text{environments}, \_ : \Rightarrow T) : \Rightarrow T$

$\text{scope}(D, X)$ executes D with the current bindings, to compute an environment ρ representing local bindings. It then executes X to compute the result, with the current bindings extended by ρ, which may shadow or hide previous bindings.

$\text{closed}(\text{scope}(\rho, X))$ ensures that X can reference only the bindings provided by ρ.

Rule $$\frac{\text{environment}(\text{map-override}(\rho_1, \rho_0)) \vdash X \longrightarrow X'}{\text{environment}(\rho_0) \vdash \text{scope}(\rho_1 : \text{environments}, X) \longrightarrow \text{scope}(\rho_1, X')}$$

Rule $\text{scope}(\_ : \text{environments}, V : T) \rightsquigarrow V$

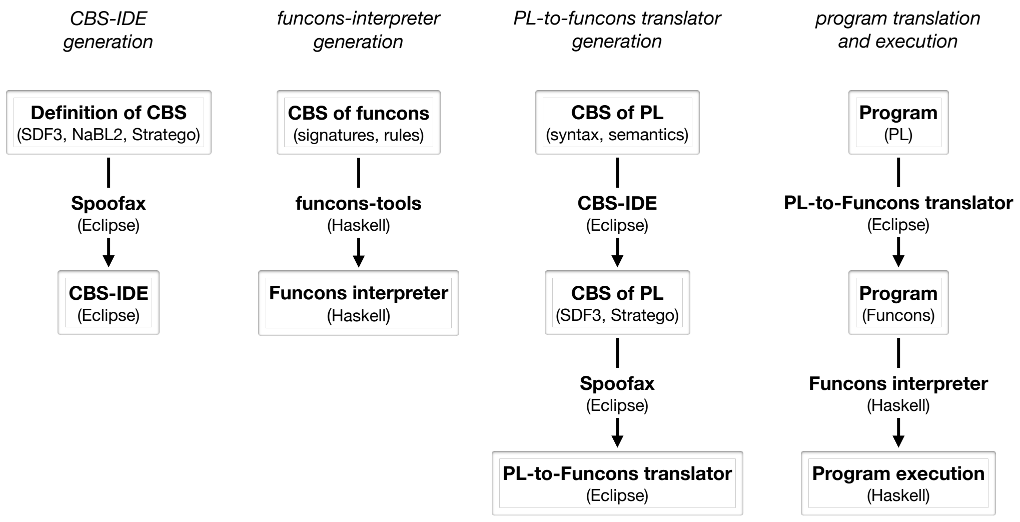

CBS-IDE generation
funcons-interpreter generation
PL-to-funcons translator generation
program translation and execution
Definition of CBS (SDF3, NaBL2, Stratego)
Spoofax (Eclipse)
CBS-IDE (Eclipse)
CBS of funcons (signatures, rules)
funcons-tools (Haskell)
Funcons interpreter (Haskell)
CBS of PL (syntax, semantics)
CBS-IDE (Eclipse)
CBS of PL (SDF3, Stratego)
Spoofax (Eclipse)
PL-to-Funcons translator (Eclipse)
Program (PL)
PL-to-Funcons translator (Eclipse)
Program (Funcons)
Funcons interpreter (Haskell)
Program execution (Haskell)

Figure 1 Generation and use of a CBS-IDE based on Spoofax and external tools [16].

The CBS language project also supports generation of parsers and translators from language specifications. It transforms the context-free grammar of a language specification in CBS to the corresponding SDF3 grammar; it transforms the equations defining the semantic functions to corresponding rewrite rules in Stratego. The resulting language project can parse programs in the specified language, then translate the programs to funcon terms.

External tools (implemented originally in Prolog, and subsequently in Haskell [28]) support evaluation of funcon terms according to the rules that define the funcons, thereby testing whether the rules specify the expected results. Moreover, the combination of these external tools with the Spoofax language project generated from a language specification allows programs in the specified language to be run according to their translation to funcons; this tests the translation rules as well as the rules from the funcon definitions. Figure 1 gives an overview of the tool chain; see reference [16] for further details. *Running the semantics* on suites of test programs, using artefacts generated automatically from the semantics, can reveal subtle errors, as well as eliminating trivial mistakes [9].

4 Generation of Hyperlinked Twins

This section explains how to provide the same name-based code navigation in web browsers as in Spoofax. It uses a toolchain that combines Spoofax with some standard applications to produce *hyperlinked twins* of CBS source files: web pages displaying the same content as the source files, but with the addition of hyperlinks from references to declarations. The web pages also add syntax highlighting, corresponding to that provided by Spoofax when browsing CBS source files locally.

The CBS language project generates hyperlinked twins of CBS source files in three formats: PLAIN, PRETTY, and PDF. For each format, Stratego rules specify a recursive traversal of the analysed AST of the source file, generating strings in the `kramdown` markup language (a variant of Markdown) [12]. The NaBL2 API for Stratego provides strategies to test whether a node of the AST is a declaration or a reference; and when name resolution has determined the declaration to which a reference refers, the API also provides the path of the file that contains the declaration.

Recall that CBS specifications may include informal text as comments. In the source files, comments are enclosed in `/*...*/`; the hyperlinked twins display comments as running text, and `kramdown` determines the layout. Embedded references to names in comments become (highlighted) hyperlinks to declarations. The formal parts of the CBS specifications are displayed as code blocks.

4.1 PLAIN format

The PLAIN format preserves the layout (indentation, line breaks) of the CBS specification by enclosing it in a pre-formatted HTML element (`<pre>`), which is rendered by browsers with a fixed-width font. References and declarations in CBS generate anchor elements (`<a>`) in HTML. Syntax highlighting is produced by HTML span elements. The static site generator Jekyll [6] renders the generated `kramdown` file (prefixed with some meta-data) on the CBS-beta website.

The PLAIN format shows how to write CBS in source files, and the correspondence between a source file and the hyperlinked web page is direct. However, CBS is based on mathematical notation from denotational and operational semantics, and publications about CBS generally display specifications with mathematical typography. The PLAIN format represents an inference rule by a sequence of dashes between the the premises and the conclusion; and it represents a labelled transition by enclosing the label in `--` and `->`. Some of the other approximations of mathematical symbols by ASCII characters are similarly indirect, as well as inelegant. The PRETTY and PDF formats address those issues.

4.2 PRETTY format

For the PRETTY format, the Spoofax language project for CBS generates LaTeX math-mode markup from the analysed ASTs of CBS source files. The `kramdown` variant of Markdown allows such blocks, but leaves their rendering to math typesetting libraries such as KaTeX [7] and MathJax [13].

To avoid dependence on low-level details of the LaTeX commands supported by KaTeX and MathJax, the author has developed CBS-LaTeX [2], a small LaTeX package for CBS specifications. When CBS is marked up using the commands defined by the package, LaTeX formatting produces mathematical typography, suitable for inclusion in published articles. CBS-LaTeX also includes KaTeX and MathJax configurations that produce similar-looking results from the same LaTeX mark-up when embedded in web pages.

Markup using CBS-LaTeX is quite low-level; this makes it easy to adjust the layout to fit the intended page width, but tedious to write. The markup generated by the CBS language project includes line breaks in places where they should enhance readability; it also respects the line breaks in formulae in the source files.

4.3 PDF format

For the PDF format, `kramdown` takes the Markdown with LaTeX math blocks used for the PRETTY format, and converts the Markdown to text-mode LaTeX. Using the CBS-LaTeX package, `pdflatex` then produces a PDF document where the layout and formatting match the rendering of the PRETTY web page in the browser.

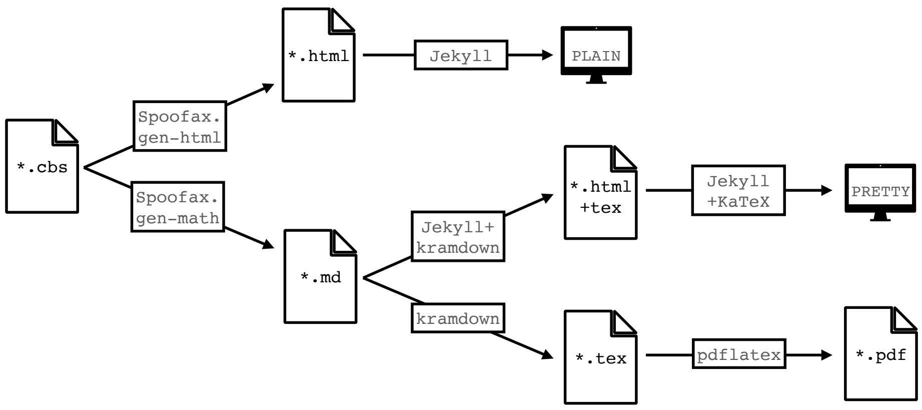


Figure 2 Generation of hyperlinked twins using Spoofax and external tools.

4.4 Offline generation

Although the definitions of funcons are to be fixed (after the finalisation of Funcons-beta) their grouping in files may change. Moreover, the informal comments that motivate and explain the funcons are not definitive, and can change. The Spoofax language project for CBS provides buttons to update the generated Markdown files when needed, and GitHub Pages automatically rebuilds the CBS-beta website when changes are pushed to the CBS-beta repository. However, Spoofax can be used offline: the application Sunshine2 [24] can open Spoofax in Eclipse, build a specific project, and execute the actions attached to buttons. Offline regeneration of all outdated files has been automated using a Makefile.

Figure 2 summarises the toolchains used to generate the PLAIN, PRETTY, and PDF hyperlinked twins of CBS source files.

5 Related Approaches

The generation of websites from Literate Agda specifications provided the initial inspiration for a hyperlinked twin of the CBS source files. In particular, the online book *Programming Language Foundations in Agda* [32] was generated from Literate Agda sources, with name references hyperlinked to definitions. Agda is an indentation-sensitive language, so the generated HTML respects layout in the same way as the PLAIN format on the CBS-beta website. Literate Agda allows informal text to be marked up in Markdown. One difference is that Agda source files make extensive use of Unicode characters, in contrast to the ASCII approximation to mathematical notation used in CBS source files. The implementation of Literate Agda uses `pandoc` to convert Markdown to HTML, analogously to how the CBS-beta website uses `kramdown`.

GitHub currently supports *search-based* code navigation for files in 10 languages: C#, CodeQL, Elixir, Go, Java, JavaScript, PHP, Python, Ruby, and TypeScript. Searching for the definition of a referenced name may return irrelevant results; similarly when searching for all references to a particular definition of a name.

GitHub also supports *precise* code navigation for Python. The name resolution is based on stack-graphs [4], which are closely related to the scope graphs used in Spoofax. Understandably, GitHub is focusing its support for precise code navigation on languages that are widely used in its repositories. The cited reference states:

> Over the coming months, we will add stack graph support for additional languages, allowing us to show precise code navigation results for them as well. Our stack-graphs library is open source and builds on the *Tree-sitter* ecosystem of parsers. We will also be publishing information on how language communities can self-serve stack graph support for their languages, should they wish to.

In principle, it should be possible to migrate the grammar for CBS from SDF3 to *Tree-sitter*, and the name resolution rules for CBS from NaBL2 to stack-graphs. But it appears that doing that would not provide code navigation in CBS repositories on GitHub.

An alternative approach would be to implement CBS support for the Language Server Protocol (which is not currently incorporated in Spoofax). Then web-enabled IDEs (e.g., Visual Studio Code [30]) would be able browse CBS repositories both locally and online. Name resolution for CBS would need to be implemented in a code indexing format such as LSIF or SCIP [23]. That might be straightforward for CBS (due to the uniqueness and global visibility of funcons in Funcons-beta) but perhaps not for the Spoofax meta-languages and the other languages that have already been specified in Spoofax.

6 Conclusion and Future Work

Spoofax has been used in toolchains to generate the PLAIN, PRETTY, and PDF hyperlinked twins of the specifications in the CBS-beta repository. It could be used in the same way for any other repository of specifications in the CBS-beta language, after setting up a Jekyll website in it.

In the CBS-beta repository, each language in Languages-beta is a separate Spoofax language project, with a symbolic link to the Funcons-beta project. On GitHub, other repositories could include Funcons-beta as a submodule, to make the funcon definitions locally available and avoid the need for inter-repository name resolution.

It should be straightforward to generate PLAIN hyperlinked twins of specifications written in the Spoofax meta-languages (SDF3, Statix, etc.), since name resolution for those languages has already been specified in Spoofax. The tree-to-string traversal is specified generically in Stratego for nodes that define or reference names; adding syntax highlighting involves inserting further HTML markup for the relevant nodes, but other nodes generate strings uniformly. To generate PRETTY and PDF twins would require transformation to combinations of Markdown and LaTeX math blocks.

The techniques presented here might be useful for online code navigation also in small programming languages (including DSLs) and specification languages when their syntax and name resolution can be easily specified in Spoofax. For other languages, the alternative techniques to support code navigation discussed in Section 5 may be preferred.

To be able to jump straight to the definition of a funcon from all references to its name is the most important feature of name-based navigation in CBS-beta specifications. Currently, the generated hyperlinked twins do not support finding all references to definitions. The list of all references to a definition is available in the analysed AST, and displayed by Spoofax when browsing the definition; it could be displayed in the same way in the PLAIN and PRETTY twins, but it is unclear how to display it in the PDF twin.

The author is planning to re-specify name resolution for CBS-beta in the Statix meta-language. Statix has an incremental implementation, which will avoid repeatedly re-analysing the Funcons-beta project while editing a language specification. This should remove the main source of inefficiency with interactive use of the current Spoofax language project for CBS-beta, in preparation for a release of the CBS-beta language project as an Eclipse plugin.

References

1 CBS-beta. A framework and meta-language for component-based specification of programming languages, accessed 2023-01-30. URL: `https://plancomps.github.io/CBS-beta/`.

2 CBS-LaTeX. A LaTeX package for CBS specifications, accessed 2023-01-30. URL: `https://plancomps.github.io/cbs-latex/`.

3 Martin Churchill, Peter D. Mosses, Neil Sculthorpe, and Paolo Torrini. Reusable components of semantic specifications. *LNCS Trans. Aspect Oriented Softw. Dev.*, 12:132–179, 2015. `doi:10.1007/978-3-662-46734-3_4`.

4 Douglas Creager. Introducing stack graphs. GitHub Blog, 2021. URL: `https://github.blog/2021-12-09-introducing-stack-graphs/`.

5 Damian Frölich and L. Thomas van Binsbergen. iCoLa: A compositional meta-language with support for incremental language development. In Bernd Fischer, Lola Burgueño, and Walter Cazzola, editors, *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2022, Auckland, New Zealand, December 6-7, 2022*, pages 202–215. ACM, 2022. `doi:10.1145/3567512.3567529`.

6 Jekyll. A static site generator, accessed 2023-01-30. URL: `https://jekyllrb.com`.

7 KaTeX. A math typesetting library for the web, accessed 2023-01-30. URL: `https://katex.org/`.

8 Lennart C. L. Kats and Eelco Visser. The Spoofax language workbench. In William R. Cook, Siobhán Clarke, and Martin C. Rinard, editors, *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*, pages 237–238. ACM, 2010. `doi:10.1145/1869542.1869592`.

9 Casey Klein, John Clements, Christos Dimoulas, Carl Eastlund, Matthias Felleisen, Matthew Flatt, Jay A. McCarthy, Jon Rafkind, Sam Tobin-Hochstadt, and Robert Bruce Findler. Run your research: on the effectiveness of lightweight mechanization. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 285–296. ACM, 2012. `doi:10.1145/2103656.2103691`.

10 Gabriël Konat, Sebastian Erdweg, and Eelco Visser. Bootstrapping domain-specific meta-languages in language workbenches. In Bernd Fischer and Ina Schaefer, editors, *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2016, Amsterdam, The Netherlands, October 31 - November 1, 2016*, pages 47–58. ACM, 2016. `doi:10.1145/2993236.2993242`.

11 Gabriël D. P. Konat, Lennart C. L. Kats, Guido Wachsmuth, and Eelco Visser. Declarative name binding and scope rules. In Krzysztof Czarnecki and Görel Hedin, editors, *Software Language Engineering, 5th International Conference, SLE 2012, Dresden, Germany, September 26-28, 2012, Revised Selected Papers*, volume 7745 of *Lecture Notes in Computer Science*, pages 311–331. Springer, 2012. `doi:10.1007/978-3-642-36089-3_18`.

12 kramdown. A library for parsing and converting a superset of Markdown, accessed 2023-01-30. URL: `https://kramdown.gettalong.org`.

13 MathJax. A JavaScript display engine for mathematics, accessed 2023-01-30. URL: `https://mathjax.org/`.

14 MetaBorg. GitHub organisation, accessed 2023-01-30. URL: `https://github.com/metaborg`.

15 Peter D. Mosses. Modular structural operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:195–228, 2004. `doi:10.1016/j.jlap.2004.03.008`.

16 Peter D. Mosses. A component-based formal language workbench. In Rosemary Monahan, Virgile Prevosto, and José Proença, editors, *Proceedings Fifth Workshop on Formal Integrated Development Environment, F-IDE@FM 2019, Porto, Portugal, 7th October 2019*, volume 310 of *EPTCS*, pages 29–34, 2019. `doi:10.4204/EPTCS.310.4`.

17 Peter D. Mosses. Software meta-language engineering and CBS. *J. Comput. Lang.*, 50:39–48, 2019. `doi:10.1016/j.jvlc.2018.11.003`.

18 Peter D. Mosses. Fundamental constructs in programming languages. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation - 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece, October 17-29, 2021, Proceedings*, volume 13036 of *Lecture Notes in Computer Science*, pages 296–321. Springer, 2021. `doi:10.1007/978-3-030-89159-6_19`.

19 Pierre Neron, Andrew P. Tolmach, Eelco Visser, and Guido Wachsmuth. A theory of name resolution. In Jan Vitek, editor, *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9032 of *Lecture Notes in Computer Science*, pages 205–231. Springer, 2015. `doi:10.1007/978-3-662-46669-8_9`.

20 Daniël A. A. Pelsmaeker, Hendrik van Antwerpen, and Eelco Visser. Towards language-parametric semantic editor services based on declarative type system specifications. In Alastair F. Donaldson, editor, *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom*, volume 134 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ECOOP.2019.26`.

21 PLanCompS: Programming language components and specifications. Home page, accessed 2023-01-30. URL: `https://plancomps.github.io`.

22 Arjen Rouvoet, Hendrik van Antwerpen, Casper Bach Poulsen, Robbert Krebbers, and Eelco Visser. Knowing when to ask: sound scheduling of name resolution in type checkers derived from declarative specifications. *Proc. ACM Program. Lang.*, 4(OOPSLA):180:1–180:28, 2020. `doi:10.1145/3428248`.

23 SCIP Code Intelligence Protocol. Sourcegraph Blog post, accessed 2023-01-30. URL: `https://about.sourcegraph.com/blog/announcing-scip`.

24 spoofax-sunshine. An application for running Spoofax, accessed 2023-01-30. URL: `https://github.com/metaborg/spoofax-sunshine`.

25 Hendrik van Antwerpen, Pierre Neron, Andrew P. Tolmach, Eelco Visser, and Guido Wachsmuth. A constraint language for static semantic analysis based on scope graphs. In Martin Erwig and Tiark Rompf, editors, *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 49–60. ACM, 2016. `doi:10.1145/2847538.2847543`.

26 Hendrik van Antwerpen, Casper Bach Poulsen, Arjen Rouvoet, and Eelco Visser. Scopes as types. *Proc. ACM Program. Lang.*, 2(OOPSLA):114:1–114:30, 2018. `doi:10.1145/3276484`.

27 Hendrik van Antwerpen and Eelco Visser. Scope states: Guarding safety of name resolution in parallel type checkers. In Anders Møller and Manu Sridharan, editors, *35th European Conference on Object-Oriented Programming, ECOOP 2021, July 11-17, 2021, Aarhus, Denmark*, volume 194 of *LIPIcs*, pages 1:1–1:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ECOOP.2021.1`.

28 L. Thomas van Binsbergen, Peter D. Mosses, and Neil Sculthorpe. Executable component-based semantics. *J. Log. Algebraic Methods Program.*, 103:184–212, 2019. `doi:10.1016/j.jlamp.2018.12.004`.

29 L. Thomas van Binsbergen, Neil Sculthorpe, and Peter D. Mosses. Tool support for component-based semantics. In Lidia Fuentes, Don S. Batory, and Krzysztof Czarnecki, editors, *Companion Proceedings of the 15th International Conference on Modularity, Málaga, Spain, March 14 - 18, 2016*, pages 8–11. ACM, 2016. `doi:10.1145/2892664.2893464`.

30 Visual Studio Code: Language server extension guide. Visual Studio Code API, accessed 2023-01-30. URL: `https://code.visualstudio.com/api/language-extensions/language-server-extension-guide`.

31 Guido Wachsmuth, Gabriël D. P. Konat, and Eelco Visser. Language design with the Spoofax language workbench. *IEEE Softw.*, 31(5):35–43, 2014. `doi:10.1109/MS.2014.100`.

32 Philip Wadler, Wen Kokke, and Jeremy G. Siek. *Programming Language Foundations in Agda*. Online book, August 2022. URL: `https://plfa.inf.ed.ac.uk/22.08/`.